Surfing Peer-to-Peer IPTV: Distributed Channel Switching

A.-M. Kermarrec[1], E. Le Merrer[1⋆], Y. Liu[23⋆] and G. Simon[2]

[1] INRIA Centre Rennes - Bretagne Atlantique, France
[2] Institut TELECOM - TELECOM Bretagne, France
[3] State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, China

Abstract. It is now common for IPTV systems attracting millions of users to be based on a peer-to-peer (P2P) architecture. In such systems, each channel is typically associated with one P2P overlay network connecting the users. This significantly enhances the user experience by relieving the source from dealing with all connections. Yet, the joining process resulting in a peer to be integrated in channel overlay usually requires a significant amount of time. As a consequence, switching from one channel to another is far to be as fast as in IPTV solutions provided by telco operators. In this paper, we tackle the issue of efficient channel switching in P2P IPTV system. This is to the best of our knowledge the first study on this topic. First, we conducted and analyzed a set of measurements of one of the most popular P2P systems (PPlive). These measurements reveal that the set of contacts that a joining peer receives from the central server are of the utmost importance in the start-up process. On those neigbors, depends the speed to acquire the first video frames to play. We then formulate the switching problem, and propose a simple distributed algorithm, as an illustration of the concept, which aims at leveraging the presence of peers in the network to fasten the switch process. The principle is that each peer maintains as neighbors peers involved in other channels, providing peers with *good* contacts upon channel switching. Finally, simulations show that our approach leads to substantial improvements on the channel switching time. As our algorithmic solution does not have any prerequisite on the overlays, it appears to be an appealing add-on for existing P2P IPTV systems.

1 Introduction

The diffusion of television over Internet, known as IPTV, has fostered a huge amount of works. Among the most studied architectures, *P2P systems* have produced not only theoretical proposals (see [16] for an overview of the main theoretical challenges) but also practical applications used by millions of users (*e.g.*, PPlive, sopcast, PPStream). For instance, the number of visitors of PPlive [1] website reached 50 millions for the opening celebration of Olympics (source:

⋆ Supported by project P2Pim@ges, of the French Media & Networks cluster

Data Center of China Internet) while the dedicated Olympic channel attracted 221 million of users in only two weeks. However, these implementations do not provide efficient channel switching features, while this is well-known as a natural TV watcher behavior. Typically, the time interval from when one new channel is selected until actual playback starts on the screen can be prohibitively long in current P2P streaming systems. More specifically, the only work that has, to the best of our knowledge, evaluated this start-up delay reports that it requires from 10 to 20 seconds to switch to a popular channel and up to 2 minutes for less popular channels [10]. Should these recent measurements highlighting the high frequency of switching system be confirmed [5,6], this could be a burden for the success of P2P IPTV systems.

In multicast-based IPTV (networks controlled by telco operators), a set of solutions has been designed to reduce the start-up delay [7] upon channel switching. They mostly consist in sending data of some *adjacent* channels along with the current channel data. Two channels C_1 and C_2 are called adjacent if when a user watches C_1, if (s)he switches channels, the probability that C_2 is chosen is high. For example, it has been shown that a user watching a sport channel, has a high probability to switch to another sport channel [5]. Thus, should the user switch from a channel to one in the adjacent channel set, the corresponding multicast traffic is received without suffering from any network delay. Several works have extended this technique in order to maximize the probability that the target channels are within the set of adjacent channels [13,3].

Yet, in P2P IPTV systems, receiving simultaneously several multimedia flows, even degraded, remains too expensive (important overhead of applicative multicast over IP, compared to lower layer multicast). However, users switching patterns being similar [5], we leverage the fact that switching channels mostly involve adjacent channels. In this paper, we make an initial step to analyze and solve the channel switching issue in P2P IPTV with a simple approach that could be potentially implemented over all existing P2P systems. The other works related with multi-channel systems, concurrent to this study, have not addressed the neighborhood discovery problem [18,19]. Our contributions are threefold.

First, in an attempt to characterize the criticity of joining a new channel for the playback delay, we measure and analyze the PPlive system [1] focusing on the so-called *bootstrap time*: the time between the reception of the P2P contacts, *i.e.*, a peerlist (a list of supposedly active peers given by a central server for a given channel) and the time at which the first video packet is received. A joining peer is expected to be able to discover new peers from this initial contact lists through request propagation in its new channel overlay. However, our measures show that in PPlive, most of the initial video content is actually provided by those contacts given by the server. Our study demonstrates their importance as well as shows that the ratio of peers effectively active in the peerlist is particularly low.

Second, we define the distributed channel switching problem and describe a simple yet efficient solution in which a peer watching a given channel also keeps some links to few peers in specific channels, typically adjacent channels. The peers with which a peer exchanges video content related with its channel are

called *overlay neighbors*. The peers maintained from other channels are called *contact peers*. When a peer x has a contact peer in the channel c, x is a *switcher* for c. Obviously a peer cannot be a switcher for all channels as the traffic generated for the maintaining of contact peers can not be neglected. Instead, peers may leverage their overlay neighbors to find switchers for more channels. Our goal is to ensure that, in a given overlay neighborhood, the number of adjacent channels covered by switchers is maximized. We show that an implementation solution of the switching responsibility distribution over a given overlay network is closely related with a (r, k)-*configuration problem* [8], a NP-hard problem. From a theoretical side, no exact solution can be computed in a reasonable time, even if one could have a global view of the system. Therefore, we provide a practical solution approximating the optimal solution. Our algorithm is simple, local, efficient and able to cope with dynamic behavior of P2P systems.

Although, this algorithm has been designed for channel switching, its applicability goes beyond. More generally, the problem addressed in this paper consists of switching from a highly connected clusters of peers to another highly connected clusters within a giant overlay network. Typically, switching from one chapter to another chapter in a P2P VoD streaming system admits a very close problem formulation, and solutions are unsurprisingly related with prefetching of most probable seeking positions [20,9].

Finally, we provide a comprehensive set of simulations. Actually, it is difficult to compare our proposal to other existing implemented solutions because, to the best of our knowledge, only centralized algorithms are used by current systems. Yet, we show that our proposal significantly improves the quality of peers that are given to a joining peer in a channel, and then in practice reduces time for this peer to get the first video packets (start-up delay reduction).

Totally, the main contributions are the following: (*i*) we measure and analyze the bootstrapping of PPlive, motivating the need for a faster process; (*ii*) we formulate the problem of distributed channel switching, and (*iii*) we propose and simulate a greedy algorithm to the distributed channel switching problem.

The rest of this paper is organized as follows. Section 2 details our measurements of start-up delays of PPlive. Section 3 formulates the channel switching problem. Section 4 presents our algorithm proposal. Section 5 reports simulation results, and Section 6 finally concludes the paper.

2 On the importance of given peer sets in PPlive

To understand the bootstrap process, and assess its importance for current popular P2P streaming systems, we conducted a measurement study of PPlive in July and August 2008. Application protocols are not publicized by PPlive, justifying a reverse-engineering by practical measurements (*active crawling* and *passive sniffing* [10]), to understand this critical phase. We focus on the bootstrapping process, *i.e.* the first two minutes of connections. Depending on channel popularity estimated by the PPlive website, we define five classes of channels: from *1-star* popularity grade (the less popular channels) to *5-star* popularity grade. In most

Channel popularity	% of responding peerlist's peers	Avg number of overlay neighbors	% of ov. neighbors $\notin$ peerlist	T_{start} (seconds)	T_{start} $\notin$ peerlist
1	2.1	4.90	34.69	9.61	25.44
2	7.0	16.40	35.98	10.73	34.49
3	10.9	23.20	32.44	11.92	49.84
4	17.2	33.75	23.70	11.21	63.27
5	16.1	30.87	21.86	9.76	47.46

Fig. 1: Measurement results for bootstrap procedure in PPlive

of experiments (more than 95%), the residential peer receives the *peerlists* from three servers. Each peerlist consists of 50 peers that are assumed to watch the same channel, in order for the joining peer to bootstrap. 20 channels are selected per class, except for the 5-star class for which 8 channels only were available.

A first set of results is reported in Fig. 1 (first 4 columns). We focus here on overlay neighbors providing at least one video packet during the first two minutes. These peers are known either through the initial peerlist, or through subsequent requests from joining peers to their neighbors. We observe that the ratio of peers that belong to the initial peerlist and actually send eventually a packet (second column) is low. This shows the low quality of peers provided through the bootstrap process. Less than 17% of peers that have been provided by servers for the bootstrap are delivering video content. For less popular channels, this ratio is even worse (2.1%). This can be explained by the fact that peers are active during a smaller duration in non popular channels, so servers' knowledge of active peers is unperfect. Presence of NATs, and overhead tradeoff for refreshing of peer information towards the servers may explain these low ratios. This is an issue that motivates the need for a dedicated strategy for improving the discovery of trustworthy peers for the bootstrap process. It can be done by contacting less peers, but preferably good and active ones for efficiency. Then, we enumerate the number of peers that deliver at least one packet during the first two minutes (third column); we observe that more peers are forwarding content in most popular channels, thus virtually providing an increased quality of service. Finally, the fourth column shows that approximately one third of peers that are actively providing content (protocol neighbors) are not given by servers in peerlists. Those results show that the initial peerlists given to a joining peer are crucial. If they are not good enough, an additional process for requesting new peers to participate is needed, therefore adding some delay for video start. Note that this need is even more stringent for low popularity channels.

A second set of results is depicted in Fig. 1 (last two columns). We measure the time T_{start} which is the average interval between getting the initial peerlist from servers and receiving the first packet. In all cases, the first packet is received from a peer in the initial peerlist. The results show that peers that are contacted are not immediately responding, and the required time to send the packet is approximately constant, around 10 seconds. The last column shows the time at which the first packet is received from a peer obtained by an additional search process (not in the initial peerlist). The results are very different depending on

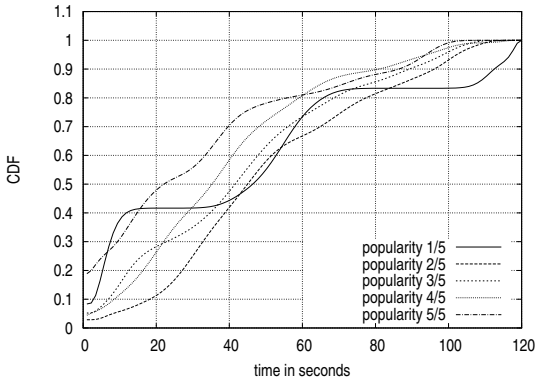

Fig. 2: CDF of the duration of video packet delivering for transient neighbors

the popularity of channels (time increases with popularity). This shows that the additional process is more critical for unpopular channels. Effectively, [10] shows that playback starts up to 2 minutes after a join for unpopular channels; here the first packet is received at best 25 seconds after the reception of peerlists, highlighting this process criticity. Contrariwise, for popular channels, first packets arrive after playback started; they help for improvement of quality of service instead of initial playback.

The last measurement is depicted in Fig. 2. We present the time interval between the first packet and the last packet received from a peer. As we measure only the first two minutes of bootstrap, peers still sending packets after this delay are not shown here. Fig. 2 shows the results under a Cumulative Distribution Function (CDF) where a point at (0.4,45) means that 40% of transient peers have sent video packets during less than 45 seconds. This measurement shows how long a peer is expected to be transient in delivery to send video packets. Obviously, the longer that time, the easiest for the joining peer is to predict the filling of its video buffer. The main observation is that the curves are almost linear. Therefore, there is absolutely no guarantee on the expected video delivery duration for transient peers (time during which a peer will send video packets can not be predicted). The only curve we can analyze more easily is for the most unpopular channel: its reveals three typical durations about 10 seconds for 40% of peers, then about 50 seconds for 40% of peers and almost two minutes for others. As the video delivery for this channel is mostly ensured by peers provided through the additional process, we may argue those peers have a behavior that can be more easily predicted (probably due to a particular choice, according to their importance for the application quality).

3 The distributed switching problem and our proposal

As previously shown, basic centralized managements of the bootstrap procedure, in particular the delivery of peerlists, may lead to a long playback delay.

Meanwhile, it has been shown in current mesh-based streaming systems [14] that some characteristics such as upload capacity for example may have a great impact on the quality of content dissemination. Therefore there is a tendency in such systems, to connect peers having close characteristics (upload capacity, latency, activity, *etc.*). We call such peers *matching peers* in the sequel.

In order for an overlay to connect peers regarding such characteristics, gossip-based *topology management systems* (see *e.g.*, [11]) consist in ensuring that all peers are eventually connected to matching peers, through an epidemic and iterative protocol. With a randomly chosen peerlist, a joining peer has low chance to be connected directly to matching peers, and thus should quickly improve neighborhood. Actually, despite some recent works in this direction (*e.g.*, [18,4]), a central server can hardly provide peers with matching peers quickly, in a scalable fashion. This calls for a new strategy. We describe our approach as follows.

Multi-channel IPTV modelization. We assume an IPTV system consisting of α channels, each channel being modeled as an overlay graph. The set of all overlay graphs is noted $\mathbb{G}$. A peer x is involved in exactly one P2P channel $g(x) \in \mathbb{G}$ among the α channels simultaneously offered by the IPTV service. In the P2P system, x cooperates with some other peers, called *overlay neighbors*, with the exclusive goal of exchanging content related to the channel $g(x)$. This small set of neighbors is noted $\Gamma^{prot}(x)$. Note that we focus in this paper on the topology problem, and do not address the higher level policies of packet exchanges between peers. Furthermore, let $d(x, y)$ denote the hop-distance of the shortest path between a peer x and another peer y in the overlay graph ($d(x, y) = \infty$ if $g(x) \neq g(y)$). For any integer $k > 0$, let the *k-neighborhood* of x be $(\Gamma^{prot}(x))_k = \{y | 0 < d(x, y) \leq k\}$, where y is called *k-neighbor* of x.

Distributed channel switching definition. We define the distributed channel switching problem has, for any peer, *(i)* the overlay change from a channel to another one, on a fully distributed fashion (*i.e.* without request to a central entity), and *(ii)* the quick matching with accurate peers in the new overlay, in order to shorten playback delay. These are the two metrics we promote and evaluate in this paper.

Our proposal. Considering the fact that the number of available channels is not likely to grow indefinitely in practice, and that channels have an unequal distribution of popularity in reality (Pareto in [5]), approaches that may want to create a *structured* overlay (*distributed hash table*, *skiplist*) connecting peers in all channels does not seem justified at the moment. Instead, we propose a light and simple mechanism, that aims at giving access to the most probable channels, thus capturing distributedly a fair amount of potential switches. To do so, besides the neighborhood that is used directly for the aim of the P2P protocol, a peer also maintains connections to peers belonging to other overlays, for channel switching purposes. We note $\Gamma^{inter}(x)^c$ the set of contact peers of x, for the channel c. Formally, a peer y is a contact peer of x means that y is not in the same overlay ($g(y) \neq g(x)$), but $x \rightarrow y$ relation exists. A peer x is asked to maintain fresh lists of contacts (dead nodes or nodes that switched are removed from contact lists), and to try to match with those contacts.

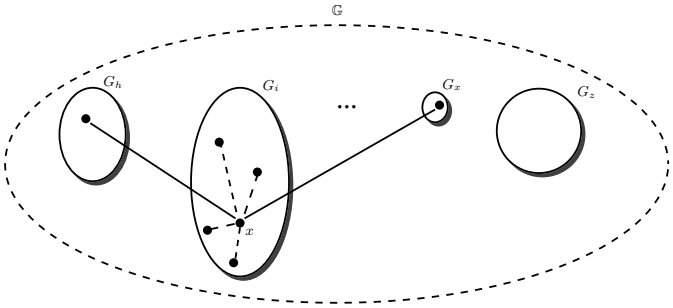

Fig. 3: An example of a resulting system $\mathbb{G}$. Node x has 4 neighbors in its current overlay G_i (dashed lines) and 2 contacts in other overlays, $\delta = 2$ (plain lines).

We associate with a peer x a set $\mathcal{C}(x)$ of overlays in $\mathbb{G}$ such that an overlay graph G belongs to $\mathcal{C}(x)$ if and only if there exists a peer y such that both $g(y) = G$ and $y \in \Gamma^{inter}(x)^G$. We say that x is a *switcher* to a channel if the overlay graph associated with this channel is in $\mathcal{C}(x)$. Because the number of channels α is expected to be large, a peer can not be a switcher to all other channels, for scalability issues (we bound by δ this maximal number). This forms a P2P overlay switching system depicted in Fig. 3.

System In a Nutshell. An *overlay switch* for a node x is the process leading to a complete change of its neighborhood $\Gamma^{prot}(x)$, to another one reflecting its move to a new chosen channel in $\mathbb{G}$. Say that a peer x in G_i wants to switch to a channel G_j; we describe now the two ways to switch overlays.

First, a peer x, has a switcher peer y (thus y has a $\Gamma^{inter}(y)^{G_j}$) in its k-neighborhood. Search could be implemented through basic expanding ring search technique, at k hops, from the requesting peer. It then only has to ask y for its contact list from G_j, to replace its protocol neighborhood, and join the targeted channel. Note that, if both overlays G_i and G_j are based on a same matching preference system (say upload capacity), this is immediately leveraged in the new overlay (as few handshakes are necessary until x discovers matching neighbors in G_j) and will speed up the convergence process.

The second scenario is simply a failure in searching distributedly a contact peer for the targeted overlay. It can either occur because of the dynamics of the overlays (switchers are not accessible at k hops), or because the chosen channel is not in the most probable ones. In such a case, the traditional centralized approach is used to get a set of contacts. Note that in IPTV, servers are mandatory, as they are in charge of pushing the content into overlays.

Main Challenges.

Allocating Contact Peers. In IPTV systems, some channels are far more popular than others [5,15]. Moreover, channels exhibit content similarities such that a participant is more likely to switch to an overlay having similar content or dealing with close activities as the one it is currently enjoying. That is, from one channel, the probability to switch to another channel is not equal for all channels. For example, paper [5] shows that 76% of switches are done in the same channel genre (*e.g.* sport, music or news). We would like every overlay to

contain switchers having few contact peers in most probable channels, so that a peer that wants to switch is likely to find a switcher in its k-neighborhood.

Ensuring Matching Contact Peers. As we have seen through measurements in PPlive, the initial peerlist given to a joining peer is crucial for a quick start-up. We use the matching property in an overlay to directly give matching (or at least close) peers in the target overlay. As previously said, our assumption is that the neighborhood reflects the matching, in other words, for two integers k_1 and k_2 with $k_1 < k_2$, if y_1 is a k_1-neighbor of x and y_2 is a k_2-neighbor of x, then it means that y_1 matches better with x than y_2. The challenge is here to maintain an accurate matching between peers, despite the overlay dynamics.

4 Protocol description

This section depicts implementation of our proposal, summarized on Algorithm 1.

Switcher creation. In order for nodes to distributedly choose for which channels they should be switchers, we refer to the notion of *domination*, well known in graph theory. A set $D \subseteq V$ of vertices in a graph $G = (V, E)$ is called a *dominating set* if, for every vertex x in V, x is either an element of D or is adjacent to an element of D. More generally, a *k-dominating set* extends this adjacency notion at k hops, at most, from a given peer x [8]. Consider that at most δ resources (here channels to be switcher for) can be allocated to a node. A *δ-configuration* is an allocation of a set of resources such that, for every resource, the vertices associated with this resource form a dominating set. The same extension as in previous definitions can state for a δ-configuration. That is, a *(δ, k)-configuration* is to allocate resources to vertices, such that no more than δ different resources are allocated to any vertex, and each vertex can access a resource associated with another vertex in less than k hops. Back to our switcher creation problem, nodes in the dominating set are responsible for keeping contacts in specific channels, and the resources are the overlays one peer has to keep in touch with.

The purpose of (δ, k)-configuration is to determine a configuration; unfortunately this decision problem has been proved to be NP-complete. That is, all optimization problems that are directly related with this decision problems are NP-hard. Thus, maximizing the number of channels that can be allocated, with a given δ and a given k is NP-hard, as well as minimizing the maximal distance and the number of switcher peers δ for a given k and a given most probable channels to cover. Therefore, an optimal solution can not reasonably be computed in a dynamic large-scale system. Instead, we propose a heuristic which enables to provide a practical and realistic alternative.

We assume that each peer joining a channel is provided by the central server or by neighbors with the list of most probable channels for switching in the same channel genre. It then checks at k hops from itself if a switcher is missing, sorted by order of importance; if so, it become a switcher for this channel (an initial peerlist is acquired from the server), with a limit of δ channels. If switchers to all most probable channels have been found, the furthest ones are chosen (l. 4-6).

Algorithm 1: A simple protocol for probabilistic switching

Initially: upon IPTV join of a node x in channel c_i
1: Arbitrarily fixed input: k (depth of switcher search), t (awaited T-Man convergence for $\Gamma^{prot}(x)$)
2: Request server c_i that returns a peerlist $\Gamma^{prot}(x)$
Matching for x in current channel:
3: T-Man resulting in matched $\Gamma^{prot}(x)$ +
4: After t cycles, periodically search for switchers at k hops: +
5: Request switchers in $\left(\Gamma^{prot}(x)\right)_k$
6: Pick a channel c_j in missing most probable, or choose the furthest one
7: Transform x into a switcher for c_j:
8: Request server c_j that returns a peer list $\Gamma^{inter}(x)^{C_j}$
9: T-Man resulting in a matched $\Gamma^{inter}(x)$
Upon switch to channel c_j:
10: Find a switcher y for c_j among $\left(\Gamma^{prot}(x)\right)_\kappa$ with increasing κ ($\kappa \leq k$)
11: $\Gamma^{inter}(y)^{C_j}$ becomes $\Gamma^{prot}(x)$ in c_j
12: Else, request server c_j that returns a peerlist $\Gamma^{prot}(x)$
13: Goto line 3

Matching through gossip. In order for peers to get connected to matching peers, according to some application predefined metric (as *e.g.* proximity, latency or bandwidth), we use a gossip based topology management similar to T-Man [11]. In this paradigm, each peer owns a value reflecting this metric. To end up with neighbors close from this value (l. 3 & 9), each peer periodically chooses its neighbor with the closest value, and exchanges with it a list of current closest neighbors and some nodes chosen randomly amongst the channel population. After each exchange, closest nodes are kept as neighbors, while furthest ones are discarded. It turns out that only a few cycles are needed to reach a near optimal peer matching, even in presence of churn [11]. The random peers needed by this protocol are provided by peer sampling protocols, that can also be based on the gossip paradigm (see *e.g.* Cyclon [17] or the Peer Sampling Service [12]).

The same matching technique as for overlay neighbors Γ^{prot} is used by switchers for contacts they keep in touch with in Γ^{inter}, except that the procedure there is not bidirectional (the switcher tries to match with peers in target overlay, but the reverse case is not true). With this matching process, once a peer wants to switch, and is able to find a switcher in its k-neighborhood, the peer contacts given by the switcher have relatively a high chance to be close from its future position in the new channel (l. 10-11).

5 Multi-channel system simulation

We evaluated our proposal using the PeerSim [2] simulator targeting large-scale and dynamic overlays. The multi-channel system we implemented as an input is based on recent application measurements conducted in [5].

Simulation Configuration.

	Centralized	Distributed
# switches	2718 (42%)	3760 (58%)
Switcher dist.	/	0.67 hops
Peerlist with ≥ 1 match	5.8%	33.3%
NPR	20%	73%

Fig. 4: Switching statistics for 100 peers

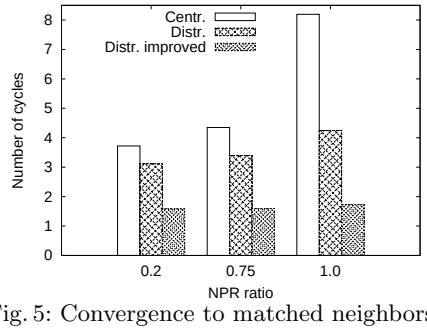

Fig. 5: Convergence to matched neighbors

The system is composed of 150 channels, classified in 13 genres (*e.g.* cine, sports, kids, docu). Channel popularity in each genre follows a Zipf distribution, that decays fast for non-popular channels; the probability to stay in the same genre during a switch is set to 76% [5,6]. Otherwise, a peer picks uniformly at random first the genre it switches to (probabilities are exposed in [5]), and then the channel in that genre. Furthermore, according to the time peers watch channels, peers get assigned a role: *surfer* (watches the channel from 0 to 45 seconds), *viewer* (46 to 3600) or *leaver* (3601 to 36000); joining and departing peers create the dynamism (or *churn*) in the system. Note that a second corresponds to one Peersim cycle execution in our simulation. After each channel switch, a peer picks a role with probability 0.6 for surfer, 0.35 for viewer and 0.05 for leaver.

Simulation parameters are set as follows: 10^5 peers in the system, an average channel-neighborhood Γ^{prot} per peer of 15 (consistent with observations made in Section 2). A peer could be a switcher for simply 1 channel ($\delta = 1$), and keeps in touch with 5 peers in this channel for Γ^{inter} (note that PPlive returns not less that three peerlists of 50 peers each). The gossip protocols (matching and sampling), for both Γ^{prot} and Γ^{inter} are executed at each cycle. After a switch, a peer waits for $t = 3$ cycles before looking for switchers, and this action is performed again every 3 cycles (to prevent a flood for search at every cycle).

Simulation Results. We run simulation 86400 cycles for an equivalent of one day of simulation. Key statistics, collected from 100 randomly selected peers, are presented on Fig. 4. Our implementation leads to 58% of distributed switches. Note that this percentage cannot be greater to the probability of switching in the same genre (here 76%), as we only provide switchers for channels in the same genre group (difference between 58% and 76% is due to churn and to the fact that only the 5 most probable channels are linked in each genre, and not all of them). The average distance between a switching peer and a switcher to the target overlay was 0.67 hops. This is due to the fact that a majority of peers are switchers themselves or could find one among their 1−neighbors.

The neighborhood perfection ratio (NPR) is a simple measure for the correctness of the matching of peers in our simulator: for a peer x, it is the number of best matches for x that are currently its neighbors, divided by the number of neighbors of x (1 then expresses that all neighbors of x are its best matches, and no better one exists in the overlay). We now observe the number of matching

peers that are given to a switching peer, by the server or by our system. In the centralized case, random selection provides only 5.8% of peerlists containing at least one matching neighbor (20% of matched contacts on average). When switches are executed distributedly, one third of the provided peerlists contain at least one good contact (with a high value of NPR of 73%), thus highlighting the improvement in providing neighbors with close characteristics.

We now look (Fig. 5) at the number of cycles needed, after a switch, to reach a given NPR. As expected, the distributed process in every case helps switching peers to reach more quickly a given ratio, as the given peerlist is closer from the switching peer than random list given by the server. We also observe that centralized switches perform relatively well: this is due to the efficiency of gossip-based topology management which converges quickly in many scenarios [11]. This motivates current streaming applications to consider gossip-based protocols for the efficiency of neighborhood management. Finally, as an important part of switching peers (33%) is immediately provided with a high quality peerlist with respect to the matching criterion (NPR of 73%), we also plot convergence time for those peers (called "Distr. improved" on Fig. 5). We observe that the time required to reach a perfect neighborhood is very short. In this case, as nearly all given neighbors are matching, the missing ones could be found in very few cycles.

This simulation study shows that a simple implementation can improve in a substantial number of cases *(i)* the quality of the initial sets provided to switching peers, thus allowing *(ii)* a fast convergence towards matching neighborhoods. Finally *(iii)*, gossip-based mechanisms appear to be an elegant and reliable solution to tackle this self-organization issue. Increased values for parameters δ, k and the number of channels linked in same genres obviously improve results. Based on observations from Section 2, this could potentially reduce significantly video start-up delays at the application level.

6 Conclusion

In this paper, we address the problem of switching from one channel to another in P2P IPTV systems, in a distributed fashion. To the best of our knowledge, no previous studies have dealt with scalable channel switching, which is a crucial issue for the next generation of multimedia delivery mechanisms over Internet. This approach shows the interest of leveraging peers' belonging to an overlay, in order to improve forthcoming switches. We believe that the simple proposed algorithm represent a first step toward the design of distributed and efficient switching mechanisms.

Acknowledgments

We would like to thank Dehao Zhang for the measurements of PPlive.

References

1. http://www.pplive.com.
2. http://peersim.sourceforge.net.
3. D. boong Lee, H. Joo, and H. Song. An effective channel control algorithm for integrated iptv services over docsis catv networks. *IEEE Transactions on Broadcasting*, 53:789–796, 2007.
4. A. Boudani, Y. Chen, and G. Simon. A quicker way to discover nearby peers. In *Proc. of the ACM CoNEXT Conference*, 2007.
5. M. Cha, P. Rodriguez, J. Crowcroft, S. Moon, and X. Amatrianin. Watching television over an ip network. In *Proc. of Usenix/ACM SIGCOMM Internet Measurement Conference (IMC)*, October 2008.
6. M. Cha, P. Rodriguez, S. Moon, and J. Crowcroft. On next-generation telco-managed p2p tv architectures. In *Proc. of International Workshop on Peer-To-Peer Systems (IPTPS)*, February 2008.
7. C. Cho, I. Han, Y. Jun, and H. Lee. Improvement of channel zapping time in iptv services using the adjacent groups join-leave method. In *Proc. of Int. Conf. on Advanced Communication Technology (ICACT)*, 2004.
8. T. W. Haynes, S. Hedetniemi, and P. Slater. *Fundamentals of domination graphs*. CRC Press, 1998.
9. Y. He, G. Shen, Y. Xiong, and L. Guan. Optimal prefetching scheme in p2p vod applications with guided seeks. *IEEE Transactions on Multimedia*, 11(1):138–151, Jan. 2009.
10. X. Hei, C. Liang, J. Liang, Y. Liu, and K. W. Ross. A measurement study of a large-scale p2p iptv system. *IEEE Transactions on Multimedia*, 9(8):1672–1687, Dec. 2007.
11. M. Jelasity and O. Babaoglu. T-man: Gossip-based overlay topology management. In *ESOA, Intl'l Work. on Engineering Self-Organising Systems*, 2005.
12. M. Jelasity, R. Guerraoui, A. marie Kermarrec, and M. V. Steen. The peer sampling service: Experimental evaluation of unstructured gossip-based implementations. In *Proc. of Middleware*, pages 79–98. Springer-Verlag, 2004.
13. J. Lee, G. Lee, S. Seok, and B. Chung. Advanced scheme to reduce iptv channel zapping time. In *Proc. of APNOMS 2007*, 2007.
14. B. Li, Y. Qu, Y. Keung, S. Xie, C. Lin, J. Liu, and X. Zhang. Inside the new coolstreaming: Principles, measurements and performance implications. In *INFOCOM 2008: Proc. of 27th IEEE Int. Conf. on Computer Communications.*, April. 2008.
15. T. Qiu, Z. Ge, S. Lee, J. Wang, Q. Zhao, and J. Xu. Modeling channel popularity dynamics in a large iptv system. In *Proc. of ACM Sigmetrics*, 2009.
16. A. Sentinelli, G. Marfia, M. Gerla, S. Tewari, and L. Kleinrock. Will IPTV Ride the Peer-to-Peer Stream? *IEEE Communications Magazine*, 45(6):86, 2007.
17. S. Voulgaris, D. Gavidia, and M. van Steen. Cyclon: Inexpensive membership management for unstructured p2p overlays. *Journal of Network and Systems Management*, 13(2):197–217, June 2005.
18. C. Wu, B. Li, and S. Zhao. Multi-channel live p2p streaming: Refocusing on servers. In *Proc. of IEEE INFOCOM*, pages 1355–1363, 2008.
19. D. Wu, Y. Liu, and K. W. Ross. Queuing network models for multi-channel p2p live streaming systems. In *Proc. of IEEE INFOCOM*, 2009.
20. C. Zheng, G. Shen, and S. Li. Distributed prefetching scheme for random seek support in peer-to-peer streaming applications. In *Proc. of the ACM workshop on Advances in peer-to-peer multimedia streaming*, 2005.